RenPar'17 / SympA'2006 / CFSE'5 / JC'2006
Canet en Roussillon, 4 au 6 octobre 2006

Utilisation du *Distributed Spanning Tree* en tant qu'*Overlay Network* pour la Recherche de Services

Sylvain Dahan, Laurant Philippe et Jean-Marc Nicod

LIFC - 16, Route de Gray - 25 030 Besançon cedex

Résumé
Le *Distributed Spanning Tree* est une topologie originale développée pour améliorer les performances des mécanismes de recherche par *flooding* en terme de vitesse de recherche et en terme de charge supportée. Ces mécanismes de recherche diffusent des messages par l'intermédiaire d'un graphe de communication. Deux types de topologie sont communément utilisés pour construire ces *overlay networks* : ce sont les graphes et les arbres. En utilisant une vision originale des arbres, il est possible de faire disparaître leurs goulets d'étranglements. Nous décrivons cette structure ainsi que deux algorithmes de parcours qui lui sont associés. Puis, les gains de performances du *Distributed Spanning Tree* par rapport aux graphes et aux arbres classiques sont soulignés à l'aide de simulations.

Mots-clés : Overlay Network, recherche de service, structure de données distribuée, simulation

---

1. Introduction

La plupart des intergiels d'application distribuée, incluant les différentes grilles de calcul, proposent des services permettant aux utilisateurs de localiser des ressources. La mise en place de ces services est indispensable pour le développement des grilles de calculs ; car ces grilles deviendraient inutilisables dans le cas où les utilisateurs seraient dans l'incapacité de trouver les ressources qui leur sont nécessaires. Deux mécanismes de recherches sont couramment utilisés pour mettre en place ces services de recherches : ce sont les annuaires et les découvertes par diffusion (*flooding*).

Les mécanismes utilisant un annuaire sont similaires aux bottins téléphoniques. Ces annuaires sont des bases de données faisant correspondre un ensemble de valeurs à une clé prédéfinie. Certains annuaires sont centralisés comme dans le cas des serveurs de nom CORBA [14]. D'autres utilisent plusieurs copies de la base de données pour offrir une meilleure qualité de service comme le fait *Universal Description, Discovery and Integration* (UDDI) [1]. Le *Domain Name Service*, quant à lui, distribue sa base de données hiérarchiquement en fonction des divers domaines d'administration. Enfin, les tables de hachage distribuées comme Chord [18] et Pastry [17] proposent de distribuer l'index sur plusieurs ordinateurs à l'aide d'algorithmes reposant sur un algorithme de routage distribué développé par Plaxton [15].

Les mécanismes usant de la découverte par diffusion sont assimilables à du porte à porte où une requête est diffusée à un ensemble de ressources dans l'espoir qu'une d'elles y réponde positivement. Cette diffusion peut se faire à l'aide d'un bus comme le font les protocoles *Ethernet Address Resolution Protocol* [16] et *Service Location Protocol* [10]. D'autres, comme le service de recherche *Service Discovery Service* [3], utilisent à cette fin un arbre de diffusion. Finalement, afin de limiter le nombre de ressources interrogées, Gnutella [12] et FastTrack [11] diffusent, par étape, un message sur un graphe de communication.

En pratique, l'utilisation de ces deux familles de mécanismes se chevauche. Pourtant, comme l'explique R. E. McGrath[13], ces deux mécanismes ont leurs spécificités qui les rendent compétitifs dans des domaines différents. La force des annuaires est de permettre d'accéder rapidement à une donnée recherchée. Leur faiblesse est le coût lié à la mise à jour de l'annuaire. Les mécanismes utilisant la diffusion n'ont, quant à eux, pas à mettre à jour de base de données au prix de recherches plus coûteuses.

L'intergiciel pour les grilles de calcul *Distributed Interactive Engineering Toolbox* (DIET) [2] recherche, pour chaque requête soumise par un utilisateur, un ensemble de ressources pouvant résoudre le problème décrit dans la dite requête. Afin d'offrir un équilibrage de charge et de proposer des ressources adaptées à un moment donné, la recherche prend en compte, en plus de la vitesse des serveurs de calcul et des logiciels qui y sont installés, la charge actuelle de ces serveurs. La charge des serveurs étant une donnée très dynamique, l'utilisation d'annuaire a été proscrite pour laisser place aux mécanismes de découverte par diffusion.
DIET, dans sa configuration multi-MA, utilise un mécanisme hybride de recherche par diffusion. Les *Master Agents* (MA) servent de points d'accès aux ressources de calcul. Chaque MA est responsable d'un ensemble de serveurs. À chaque requête d'un client, le MA diffuse la requête à l'aide d'un arbre de diffusion à l'ensemble des serveurs de calculs possédant les logiciels nécessaires à la résolution du problème décrit par le client [9]. Cependant, l'interrogation de l'ensemble des ressources pouvant répondre à un problème n'est pas extensible. C'est pour cela qu'un MA se charge d'un nombre limité de serveurs : en pratique, il s'occupera de l'ensemble des ressources d'un cluster ou de l'ensemble des ressources d'un petit centre de calcul. Afin d'étendre le nombre de ressources disponibles, DIET permet de connecter plusieurs MAs à l'aide d'un graphe de communication. Dans cette configuration, appelée multi-MA, lorsqu'un MA ne trouve pas de ressource adéquate, il transmet la requête à d'autres MA en utilisant un algorithme similaire à celui utilisé par Gnutella [7].
Afin de limiter le nombre de messages échangés sur le graphe de communication, et ainsi permettre de supporter une plus grande charge de recherche, nous avons développé une topologie adaptée à notre cas. Deux topologies sont couramment utilisées pour créer des graphes de communication sur Internet (aussi nommés *overlay network*). Ce sont les graphes et les arbres (Nous utilisons le terme de graphe pour désigner un graphe quelquonque non-directionnel). Jusqu'à maintenant, les arbres créés dans le cadre des graphes de communication sont bi-directionnel et sont construits de sorte que chaque ordinateur soit : soit une feuille qui reçoit des messages, soit un nœud intermédiaire qui reçoit et retransmet des messages. Il en découle un inconvénient typique des arbres de communication : les nœuds intermédiaires sont des goulets d'étranglement. Nous avons développé une nouvelle structure d'arbre de communication permettant de distribuer le rôle des nœuds intermédiaires sur l'ensemble des ordinateurs. Comme le rôle des nœuds intermédiaires est réparti sur l'ensemble des ordinateurs, les nœuds intermédiaires ne sont plus des goulets d'étranglement. Nous avons appelé cette structure le *Distributed Spanning Tree* (DST) [8, 4].
Cet article fournit une description des caractéristiques du DST. Pour cela, il commence par décrire la structure du DST. Ensuite, il explique comment cette structure peut-être parcourue et se termine en comparant les performances du DST face aux arbres et aux graphes pour différentes échelles de population à l'aide de simulations.

2. Structure du *Distributed Spanning Tree*

Imaginons que nous souhaitons relier un ensemble de 1 000 ordinateurs à l'aide d'un DST. Pour cela, nous créons 100 groupes de 10 ordinateurs que l'on va appeler des groupes de 10. Puis, nous regroupons les groupes de 10 pour former 10 groupes de 100. Et finalement, nous regroupons les 10 groupes de 100 en un groupe de 1 000.
Pour former un arbre, il serait possible de choisir un ordinateur de chaque groupe de 10 et de le transformer en nœud père (appelé père de 10) qui aura pour rôle de transmettre les messages aux 9 autres ordinateurs. De même, on pourrait transformer un ordinateur de chaque groupe de 100 en un nœud qui sera le père de 10 pères de 10 précédemment créés, et que l'on appellera père de 100, et ainsi de suite. Le problème, c'est que nous nous retrouvons avec un arbre de communication classique et leurs goulets d'étranglement.
Maintenant, il est tout à fait envisageable, que les ordinateurs d'un groupe de 10 se connaissent entre eux. Dans ce cas, avoir un père de 10 devient inutile car n'importe quel ordinateur du groupe de 10 est capable de prendre le rôle du père de 10 qui est de transférer des messages aux 9 autres ordinateurs. De même, il est tout à fait envisageable que tous les ordinateurs d'un groupe de 100 connaissent un ordinateur de chaque groupe de 10 composants le groupe de 100. Dans ce cas, le père de 100 devient inutile car n'importe quel ordinateur du groupe de 100 est capable de transférer le message à un ordinateur de

chaque groupe de 10 qui agira à leur tour en tant que père de 10, et ainsi de suite. Que tous les ordinateurs jouent à la fois le rôle de feuille et de père pour chaque étage de l'arbre, tel est l'idée fondatrice du DST.
L'idée centrale du DST exposée, décrivons concrètement la structure du DST. Sur un DST, tout ordinateur est obligatoirement une feuille du DST et vice-versa. La figure 1 décrit un ensemble de feuilles.

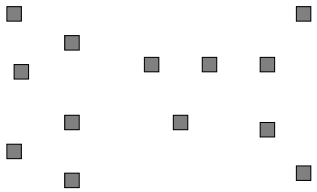


FIGURE 1 – Représentation des feuilles d'un DST.

Ces feuilles sont rassemblées pour former des petits groupes. La taille de ces groupes est bornée afin de fournir un équilibre grossier entre les différents groupes. Pour chaque groupe, un graphe complet de feuilles est formé tel que décrit par la figure 2. Ces graphes complets portent le nom de nœuds intermédiaires de niveau 1. Dans le cadre du DST, le rôle des nœuds intermédiaires est de transmettre les messages à leurs fils. Comme tous les ordinateurs d'un groupe se connaissent les uns les autres, chacun d'eux peut prendre le rôle de leur père à tout moment. Pour cette raison, et par effet d'abstraction, les graphes complets créés sont considérés comme les pères des ordinateurs qu'ils contiennent.

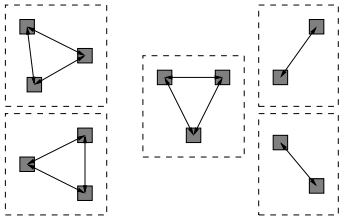
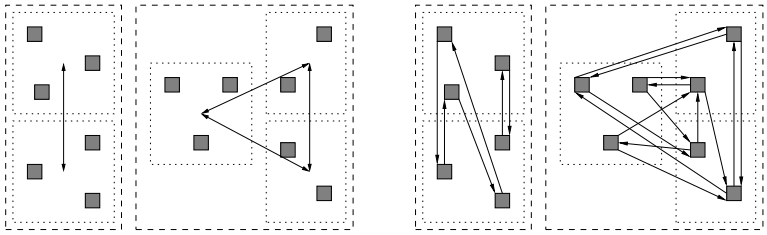


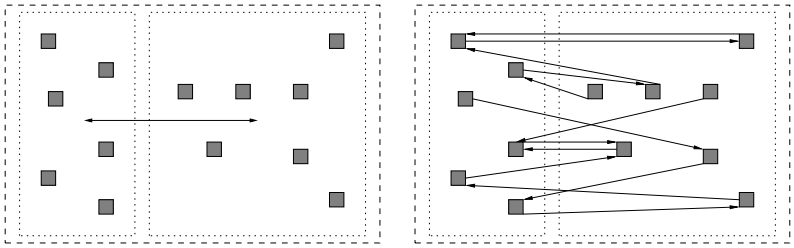
FIGURE 2 – Représentation des nœuds formant le premier étage du DST.

Les nœuds intermédiaires de niveau 1 sont regroupés en un ensemble de petits groupes. Les nœuds de niveau 1 forment à leur tour des graphes complets au sein de leur groupe comme représenté dans la figure 3.a. Ces graphes complets sont appelés nœuds intermédiaires de niveau 2. Les liens reliant deux nœuds intermédiaires sont implémentés de la manière suivante : pour tout arc reliant un nœud A à un nœud B, tous les ordinateurs intégrés au nœud A doivent avoir un lien pointant sur un ordinateur intégré au nœud B. Comme le montre la figure 3.b, l'ensemble des ordinateurs peut être utilisé comme sommets d'arrivés des arcs, ce qui permet de répartir la charge d'un nœud intermédiaire sur l'ensemble des ordinateurs qui le compose.
Puis, les nœuds intermédiaires de niveau 2 sont regroupés à l'aide de graphes complets pour former les nœuds de niveau 3 (voir figure 4.a et 4.b), et ainsi de suite. La construction s'achève lorsqu'un niveau ne possédant qu'un seul nœud intermédiaire est créé. Ce nœud est alors appelé racine du DST.



a. Représentation abstraite b. Représentation détaillée

FIGURE 3 – Représentation des nœuds formant le deuxième étage du DST.



a. Représentation abstraite b. Représentation détaillée

FIGURE 4 – Représentation des nœuds formant le troisième étage du DST.

3. Parcours d'un DST

La structure étant énoncée, nous allons décrire deux algorithmes de parcours du DST. Le premier algorithme permet de réaliser un parcours en parallèle en profondeur. Le deuxième algorithme, quant à lui, permet de réaliser des parcours aux caractéristiques similaires à celui utilisé par Gnutella.

3.1. Parcours en profondeur

D'un point de vue abstrait, les parcours en profondeur des DST sont similaires au parcours d'un arbre classique. Le nœud racine transmet un message à ses fils, qui le retransmettent à leurs fils et ainsi de suite jusqu'à ce que le message arrive au niveau des feuilles.
La figure 5 montre un exemple de parcours de DST en profondeur en trois étapes. Comme pour un arbre classique, lors de la première étape, la racine transmet un message à ses fils. Un parcours est toujours initié par un ordinateur. D'après la structure du DST, cet ordinateur fait obligatoirement partit du nœud racine. Il va donc agir en tant que nœud racine et transmettre le message aux fils de la racine. En pratique, pour chaque fils de la racine, il enverra le message à l'ordinateur qu'il connaît appartenant à ce fils. De plus, afin d'économiser un message, il s'utilise comme destinataire du message adressé au fils dont il fait partit.

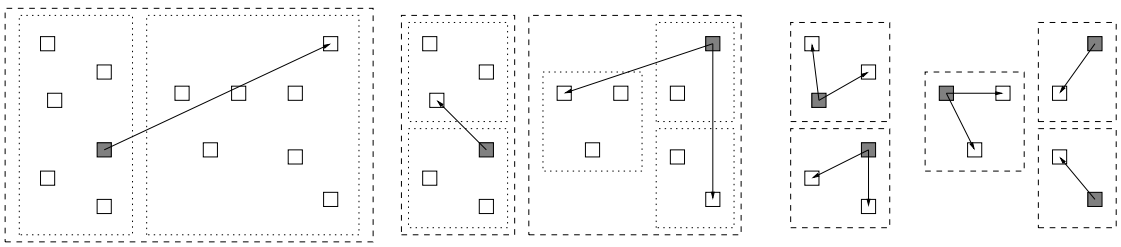


FIGURE 5 – Exemple de parcours en profondeur.

Une fois que les fils de la racine d'un arbre ont reçu le message, ils le retransmettront à leurs fils respectifs. Ainsi, les ordinateurs ayant reçu le message à l'étape précédente vont jouer le rôle des fils de la racine et transmettre le message à leurs fils respectifs. Pour cela, ils enverront le message aux ordinateurs qu'ils connaissent faisant chacun partit d'un des nœuds fils devant recevoir le message, et ainsi de suite.

3.2. Parcours par vagues successives

L'algorithme de parcours de graphe de Gnutella et d'autres protocoles de recherche paire-à-paire se fait par étapes successives. Dans une première étape, le message est transmis à tous les nœuds situés à une distance un du nœud initiateur. Puis, ce sont tous les nœuds situés à une distance deux, puis trois et ainsi de suite. L'intérêt de cet algorithme réside dans le nombre de nœuds interrogés qui croit de manière exponentielle.

Il est possible d'utiliser le parcours en profondeur décrit précédemment pour obtenir le même effet. Pourtant, ce parcours n'est pas optimal, car comme avec l'algorithme de Gnutella, certains nœuds sont contactés plus d'une fois. Par exemple, si une recherche nécessite un parcours de profondeur deux, la racine contacte ses fils lors de la première étape. Puis, elle contact ses fils une seconde fois pour transmettre le message à ces petits fils. Ainsi, chacun des fils de la racine est contacté deux fois.

La structure originale du DST lui permet de réaliser le même type de recherche sans pour autant nécessiter de contacter plusieurs fois un même ordinateur. Pour cela, lors de la première étape, l'ordinateur initiateur du parcours prend le rôle de nœud intermédiaire de niveau 1 et interroge les autres ordinateurs formant son groupe de niveau 1 (voir figure 6). Puis, l'ordinateur prend le rôle de son nœud intermédiaire de niveau 2 et diffuse un message le long de ses fils tout en évitant de le rediffuser sur le fils qui a été parcouru durant la première étape. Puis il prend le rôle du nœud intermédiaire de niveau 3 et ainsi de suite.

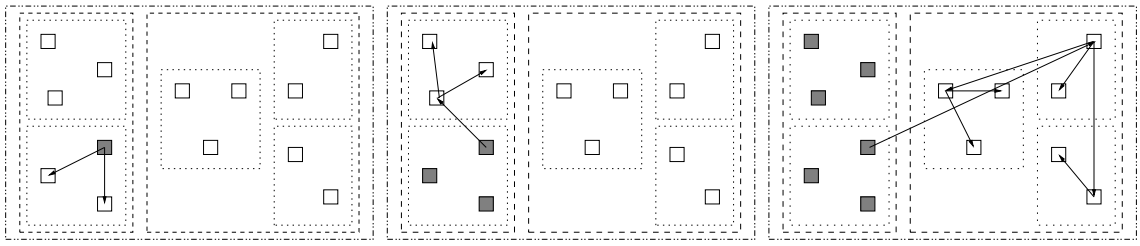


FIGURE 6 – Exemple de parcours par vagues successives.

Ainsi, à l'aide de ce parcours, le DST interroge lui aussi un nombre d'ordinateur croissant de manière exponentielle. Mais contrairement aux autres algorithmes présentés, il ne nécessite que $n - 1$ messages pour interroger n ordinateurs.

4. Études de performances par simulation

En théorie, en permettant de contacter l'ensemble des ordinateurs avec $n-1$ messages tout en distribuant la charge des nœuds intermédiaires sur l'ensemble des ordinateurs, le DST permet de meilleures performances de découverte par diffusion en terme de vitesse de recherche et en terme de charge supportée. Afin de tester cette affirmation, plusieurs simulations ont été réalisées. Ces simulations ont eu pour rôle de vérifier si la charge de recherche est bien répartie entre les différents ordinateurs et si le temps de recherches est satisfaisant.

Le simulateur développé [6] simule Internet par une topologie en étoile afin de pourvoir étudier la charge des liens reliant les ordinateurs à Internet et de faire abstraction, dans un premier temps, des mécanismes de routage d'Internet. Tous les ordinateurs sont reliés au centre de l'étoile à l'aide d'un lien à 8 Mbit/s. Les messages échangés font 1 ko et prennent donc 2 millisecondes pour être transféré d'un ordinateur à un autre si aucun autre message transite sur le réseau.

Les résultats exposés, relate la charge supportée pour des graphes de communication de 10 et 10 000 ordinateurs. Pour chaque taille de population, nous simulons des recherches par diffusion d'élément x_i. Chaque élément x_i a une probabilité de 10 % de se trouver sur chacun des ordinateurs. Pour chaque taille de population, les recherches ont été simulées sur trois types de topologies : des arbres d'arité 5, le graphe de degré 5 et des DST ayant 5 éléments par groupe. L'algorithme de Gnutella est utilisé par les recherches sur les arbres et les graphes. L'algorithme de parcours par vagues successives vue précédemment est utilisé pour réaliser les recherches sur le DST.
La figure 7 fournit les résultats obtenus pour une population de 10 ordinateurs. Les abscisses correspondent au nombre de requêtes émises par l'ensemble des ordinateurs en l'espace d'une seconde. Les ordonnées correspondent à la moyenne du temps de toutes les recherches.

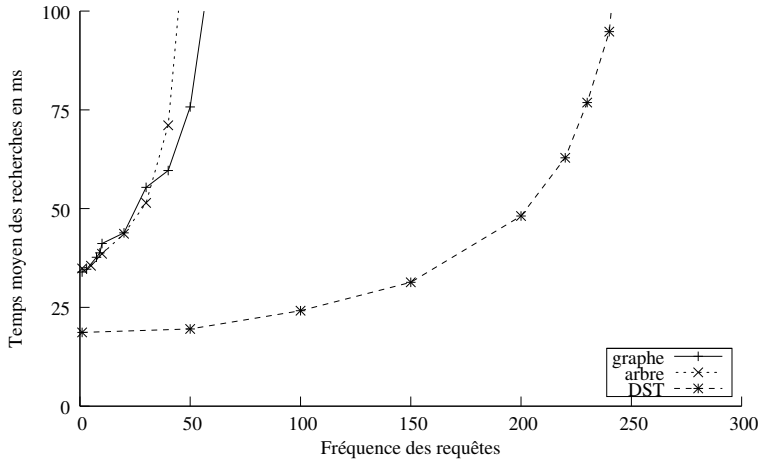


FIGURE 7 – Performances pour une population de 10 ordinateurs.

On observe clairement, que plus la fréquence des requêtes est élevée, plus le temps de recherche est important et cela pour les trois topologies. Ce comportement normal indique que la charge du système influe sur la vitesse de traitement. Les trois courbes se terminent en une asymptote indiquant la surcharge du système.
Il est intéressant de remarquer que l'arbre et le graphe ont des performances similaires. Bien que l'arbre utilise moins de messages, ces nœuds intermédiaires gèrent une grande partie du trafic et forment des nœuds de contentions. Le graphe envoie un plus grand nombre de messages et c'est l'ensemble des nœuds qui gèrent un trafic similaire à celui des nœuds de contentions de l'arbre, ce qui explique des performances similaires. Le DST quant à lui, utilise un nombre de messages légèrement inférieur à celui des arbres (voir 3.2) tout en répartissant la charge sur l'ensemble des nœuds. Il en résulte de meilleures performances en terme de vitesse de recherche et en terme de charge supportée.
La figure 8 présente les résultats obtenus pour une population de 10 000 ordinateurs. Dû aux goulets d'étranglement des arbres, les performances de l'arbre sont très mauvaises par rapport à celles du graphe et celles du DST. Les graphes quant à eux ont des performances presque aussi bonnes que le DST pour la raison suivante : ayant une probabilité de 10 % de trouver l'élément recherché sur chacun des serveurs, les recherches sont achevées en moins de 4 hops. Ayant 10 000 nœuds, il est rare qu'un nœud soit contacté plus d'une fois pour une même requête. Nous avons donc un nombre de messages échangés à peu près optimal comme le fait l'arbre ou le DST. Sauf que contrairement à l'arbre, le graphe n'a pas de goulet d'étranglement.

5. Conclusion

En distribuant le rôle des nœuds intermédiaires à travers l'ensemble des ordinateurs, le *Distributed Spanning Tree* est capable de répartir la charge de recherche sur l'ensemble des ordinateurs tout en utilisant

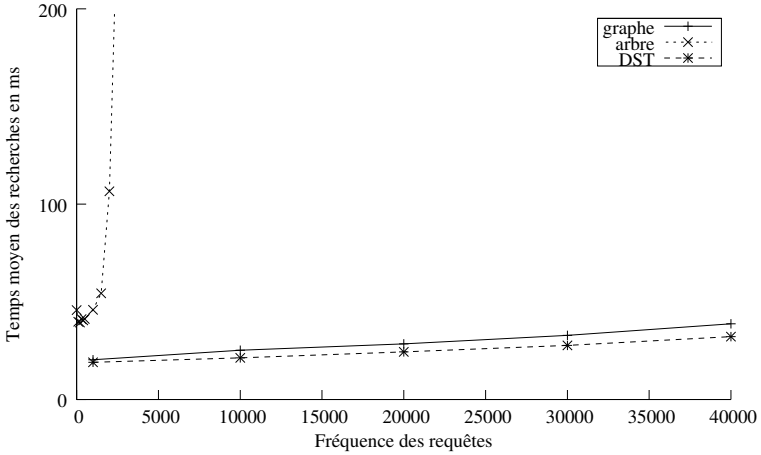


FIGURE 8 – Performances pour une population de 10 000 ordinateurs.

un nombre de messages optimal. Ce qui lui permet d'utiliser toute la largeur du réseau tout en limitant le nombre de messages échangés. Son utilisation dans le cadre de recherche par inondation implique de meilleures performances en terme de vitesse de recherche et en terme de charge supportée par rapport aux graphes et aux arbres classiques.

Les résultats présentés dans cet article ainsi que ceux présentés dans [5] indiquent que les simulations effectuées soutiennent cette hypothèse pour des graphes de communication de 10, 100, 1 000, et 10 000 ordinateurs pour des probabilités de trouver un élément sur chaque serveur variant de 0 % à 33 %. Les simulations indiquent que les graphes sont équivalents ou plus performants que les arbres en fonction de l'échelle. Les gains en terme de performances du DST ne sont pas constants par rapport à ceux des graphes. Ces gains dépendent essentiellement du pourcentage de nœuds interrogés. Plus le nombre de nœuds interrogés est élevé, plus le pourcentage qu'un nœud d'un graphe reçoive un message plusieurs fois est élevé, et plus les gains du DST par rapport aux graphes sont importants.

Dans l'attente de tests en conditions réelles, ces premiers résultats permettent de confirmer que le DST est une structure extensible dans le cadre de la recherche de service. Comparé aux graphes, le DST est une structure qui est plus complexe et nécessite donc plus d'effort pour la maintenir valide lors de l'arrivée et du départ de nœuds. Pour cette raison, nous déconseillons sont utilisations dans les architectures très dynamiques tel les programmes d'échanges de fichiers paire-à-paire. Par contre, les architectures pour lesquelles les performances sont primordiales telles que les grilles de calculs peuvent tirer pleinement bénéfice du DST.

Bibliographie

1. D. Bryan, V. Draluk, D. Ehnebuske, R. Glover, A. Hately, Y. Leng Husband, A. Karp, K. Kibakura, C. Kurt, J. Lancell, S. Lee, S. MacRoibeaird, A. T. Manes, B. McKee, J. Munter, T. Nordan, C. Reeves, D. Rogers, C. Tomlinson, C. Tosun, C. von Reigen, and P. Yendluri. UDDI version 2.04 API specification. Standard, UDDI Committee Specification, July 2002.
2. E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. A scalable approach to network enabled servers. In B. Monien and R. Feldmann, editors, *Proceedings of the 8th International EuroPar Conference*, volume 2400 of *LNCS*, pages 907–910, Paderborn, Germany, August 2002. Springer-Verlag.
3. S. E. Czerwinski, B. Y. Zhao, E. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proceedings of the fifth international conference on Mobile co mputing and networking*, pages 24 – 35, Seattle, Aug 1999. ACM Press.
4. S. Dahan. Distributed Spanning Tree algorithms for large scale traversals. In *Procedings of the11th International Conference on Parallel and Distributed Systems, ICPADS 2005*, volume 1, pages 453–459, Fukuoka, Japan, July 2005. IEEE Computer Society.

5. S. Dahan. *Mécanismes de Recherche de Services Extensibles pour les Environnements de Grilles de Calcul*. PhD thesis, U.F.R. des Sciences et Techniques de l'Université de Franche-Comté, December 2005.
6. S. Dahan. Simulator and data used for the rempar'07 article, 2006. `http://lifc.univ-fcomte.fr/~dahan/dst/rempar07-sim.tar.gz`.
7. S. Dahan, J.-M. Nicod, and L. Philippe. Scalability in a GRID server discovery mechanism. In *10th IEEE Int. Workshop on Future Trends of Distributed Computing Systems, FTDCS 2004*, pages 46–51, Suzhou, China, May 2004. IEEE Press.
8. S. Dahan, J.-M. Nicod, and L. Philippe. The Distributed Spanning Tree : A scalable interconnection topology for efficient and equitable traversal. In *Proceedings of CCGrid 2005 Workshop Global and Peer-2-Peer Computing*, Cardiff, UK, May 2005. IEEE Press. CD-ROM.
9. F. Desprez, M. Quinson, and F. Suter. Dynamic Performance Forecasting for Network-Enabled Servers in a Heterogeneous Environment. In H.R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, volume III, pages 1421–1427, Las Vegas, June 2001. CSREA Press. ISBN : 1-892512-69-6.
10. E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. Request for Comments 2608, The Internet Engineering Task Force, June 1999.
11. T. Hargreaves. The FastTrack protocol. July 2004.
12. T. Klingberg and R. Manfredi. Gnutella 0.6. RFC draft, June 2002. `http://groups.yahoo.com/group/the_gdf/files/Development/`.
13. R. E. McGrath. Discovery and its discontents : Discovery protocols for ubiquitous computing. Technical Report UIUCDCS-R-99-2132, Department of Computer Science University of Illinois, Champaign, IL, USA, April 2000.
14. OMG, http ://www.omg.org. *Trading Object Service Specification*, 97. orbos/97-07-26.
15. C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symp. on Parallel Algo. and Arch.*, pages 311–320, 1997.
16. D. C. Plummer. An ethernet address resolution protocol –or– converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. Request for Comments 826, The Internet Engineering Task Force, November 1982.
17. A. Rowstron and P. Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *LNCS*, 2218 :329–350, 2001.
18. I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. Frans Kaashoek, F. Dabek, and H. Ballakrishnan. Chord, a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Networking*, 11(1) :17–32, February 2003.